Rendu 2 : Evaluation des requettes en étoiles - NOSQL

TRINQUART Matthieu et SANCHEZ Martin

Dictionnary

Le principe du dictionnaire est d'associer chaque information de la BDD à un entier ce qui simplifiera le stockage des données (on stockera un entier et non un String) et rendra le programme plus performant

Exemple de dictionnaire :
0,http ://db.uwaterloo.ca/ galuc/wsdbm/User10
1,"7707564"
2,http ://db.uwaterloo.ca/ galuc/wsdbm/User11
3,http ://schema.org/telephone
4,"2465721"
5,"1114154"
6,http ://db.uwaterloo.ca/ galuc/wsdbm/User12

Pour stocker le dictionnaire nous utilisons deux hashmap une qui va permettre d'associer un String vers un Interger et l'autre un Integer vers un String ce qui permettra de décoder un Integer en String en O(1) et encoder un String vers un Integer dans la même complexité. Cela rend donc le programme plus performant.

```
1 HashMap<Integer, String> reverseDico;
2 HashMap<String, Integer> Dico;
```

Nous stockons le dictionnaire dans un csv ce qui rend les données persistantes.

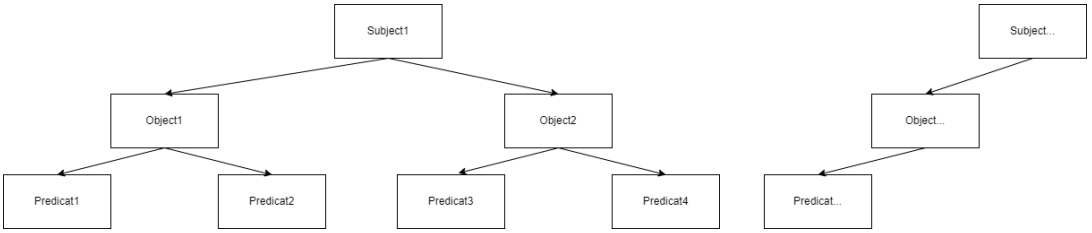

FIGURE 1 – Arbre SOP

Index

Le but de l'index est de créer 6 arbres qui SOP,SPO,PSO,POS,OSP,OPS qui représenteront les données et pour récupérer des prédicats en fonction d'un object et d'un subject il suffira de parcourir l'arbre SOP ou OSP et de retourner toutes les feuilles de ces arbres pour récupérer les données.

Tous les arbres sont représentés comme des HashMap<Integer, HashMap<Integer, ArrayList<Integer>> les Integer étant l'entier associer à un String dans le dictionnaire pour que le parcours de l'arbre soit plus optimisé (la recherche d'une hashmap étant en O(1))et les feuilles de l'arbre sont représentés comme des ArrayList<Integer> représente elle les feuilles de l'arbre.

```
1 private HashMap<Integer, HashMap<Integer, ArrayList<Integer>>> spo ;
2 private HashMap<Integer, HashMap<Integer, ArrayList<Integer>>> sop ;
3 private HashMap<Integer, HashMap<Integer, ArrayList<Integer>>> pso ;
4 private HashMap<Integer, HashMap<Integer, ArrayList<Integer>>> pos ;
5 private HashMap<Integer, HashMap<Integer, ArrayList<Integer>>> osp ;
6 private HashMap<Integer, HashMap<Integer, ArrayList<Integer>>> ops ;
```

Evaluation des requetes en étoiles

Nous avons une fonction getFromMap dans index qui prend en paramètre une hashmap (qui correspond à 1 des 6 arbres syntaxique de index) et qui parcour cette arbre en fonction des info1 et info2 qui sont soit des object, predicat ou subject

```
1 private ArrayList<Integer> getFromMap(HashMap<Integer, HashMap<Integer,
      ArrayList<Integer>>> map, String info1, String info2) {
2     ArrayList<Integer> ret = new ArrayList<>() ;
3     int i1 = Dictonnary.getInstance().encode(info1) ;
4     int i2 = Dictonnary.getInstance().encode(info2) ;
```

```
5       if(i1!=-1 && i2 != -1){
6           if (map.get(i1) != null) {
7               if (map.get(i1).get(i2)  != null) {
8                   ret.addAll(map.get(i1).get(i2)) ;
9               }
10          }
11      }
12      return ret;
13  }
```

Nous avons une fonction getFromPOS(String predicat, String object) qui prend en paramètre un object et un predicat qui va utiliser la fonction getFromMap pour parcrourir l'arbre syntaxique POS.

```
1  public ArrayList<Integer> getFromPOS(String predicat, String object) {
2      ArrayList<Integer> ret = new ArrayList<>(getFromMap(pos, predicat,
       object)) ;
3      return ret;
4  }
```

Cette fonction permet de récupérer les valeurs subject, predicat et object du StatementPattern et utilise la fonction getFromPOS de Index pour retourner le résultat de la requèt

```
1  private static ArrayList<Integer> RequestResult(StatementPattern sp) {
2      Value s = sp.getSubjectVar().getValue();
3      Value p = sp.getPredicateVar().getValue();
4      Value o = sp.getObjectVar().getValue();
5      return new ArrayList<>(Index.getInstance().getFromPOS(p.toString(), o.
       toString()));
6      }
```

Cette fonction prend en paramètre 2 ArrayList d'Integer qui représente à 2 reponses de requèt et fait l'intersection pour récupèrer l'intersection entre les deux resultat (ce qui correspond à la requèt en étoile)

```
1  private static ArrayList<Integer> intersect(ArrayList<Integer> a1,
       ArrayList<Integer> a2) {
2      ArrayList<Integer> ret = new ArrayList<>();
```

```
3       for(Integer i : a1) { if (a2.contains(i)) { ret.add(i) ; } }
4       return ret;
5   }
```

Pour la lecture des requettes nous avons utilisé la fonction processAQuery(ParsedQuery query) donné dans le main initial. On recupere une ParsedQuery, un objet qui représente une requette qu'on evalue grace à notre fonction RequestResult qui va retourner une ArrayList de String qui correspond au résultats de la requèt. On fait ensuite l'intersection entre le résultat courant (optenu par les précédente requèt) et le resultat de la request qui vient d'être exécuter.

Il retourne ensuite la résultat des requèts.

Nous affichons ensuite le résultat en décodant le résultat grace à la fonction de Dictionnary.

```
1       public static ArrayList<Integer> evaluateStarRequest(ParsedQuery query)
        {
2           ArrayList<Integer> results = new ArrayList<>() ;
3           List<StatementPattern> patterns = StatementPatternCollector.process
        (query.getTupleExpr());
4           for(StatementPattern sp : patterns) {
5               if (results.isEmpty()) { results.addAll(RequestResult(sp)) ;}
6               else { results = EvaluateRequest.intersect(results,
        RequestResult(sp)) ; }
7           }
8           return results ;
9       }
```

Comment pourrait on evaluer nimporte quelle requette

Nous avons pris le temps de reflechir a une facon de resoudre toutes les requettes en plus des requettes en etoile

Pour cela nous avons pense creer un objet 'Result' qui serai le resultat d'une requette.

```
1   public class Result {
2       private Integer s, p, o;
3       public Result(Integer s, Integer p, Integer o) {
4           this.s = s; this.p = p; this.o = o;
```

```
5         }
```

Pour cela nous avons fait une classe Result contenant le subject, predicat et object. Il suffira de regarder quelle de ces 3 valeurs est null et appeler getFromMap avec diférent arbre (SOP,PSO,etc..) en fonction de la valeur null de s, p, o.